Refine Search

Search Results -

Term	Documents
POINTER	108532
POINTERS	38729
DATA	2783090
DATUM	25392
STRUCTURE	3197151
STRUCTURES	793706
RECORD	546464
RECORDS	187406
ADDRESS	584186
ADDRESSES	245363
BLOCK	2049385
((POINTER OR DATA STRUCTURE OR RECORD OR STRUCTURE OR ADDRESS) SAME (BLOCK ADJ2 SIZE) SAME ((SEGMENT ADJ2 SIZE) OR (NUMBER ADJ3 BLOCKS ADJ3 SEGMENT)) SAME (OFFSET OR FINGER)).PGPB,USPT,EPAB,JPAB,DWPI,TDBD.	2

There are more results than shown above. Click here to view the entire set.

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L7

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Saturday, February 21, 2004 Printable Copy Create Case

Set		Set
h	e b b cg b e e ch	

Name side by side	Query	Hit Count	Name result set
	DB=PGPB,USPT,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
L7	(pointer or data structure or record or structure or address) same (block adj2 size) same ((segment adj2 size) or (number adj3 blocks adj3 segment)) same (offset or finger)	2	L7
L6	translate adj5 capability adj5 (virtual adj2 address)	3	L6
L5	translate same capability same (virtual adj2 address)	68	L5
L4	(segment or object) same permission same address same capability	15	L4
L3	L2 and l1	195	L3
L2	memory same segment	35675	L2
L1	capability adj2 (address or addressing)	1316	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

L17: Entry 10 of 12 File: TDBD Jan 1, 1982

TDB-ACC-NO: NN82014401

DISCLOSURE TITLE: Method of Revoking a Capability Containing a Pointer Type Identifier Without Accessing the Capability. January 1982.

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, January 1982, US

VOLUME NUMBER: 24
ISSUE NUMBER: 8
PAGE NUMBER: 4401 - 4403

PUBLICATION-DATE: January 1, 1982 (19820101)

CROSS REFERENCE: 0018-8689-24-8-4401

DISCLOSURE TEXT:

3p. A capability is an unforgeable, though possibly copyable, data elementt that is given to a process to enable the process to access an object that the capability identifies. - The identifier in a capability is normally either pointer type or a unique code. A pointer-type identifier is either the direct address or some kind of simple indirect address of the identified object. Thus, it maps efficiently to the address of the identified object. A unique code maps less efficiently to the address of the identified object through the use of a hashing table. - It is sometimes necessary to revoke a capability either because the relationship between the owning process and the identified object has changed or because the object is to be destroyed. A capability containing a pointer-type identifier normally is revoked by accessing the capability and setting it invalid. (If the identifier is indirect, an alternative is to set invalid the entry in the intermediate mapping table. This is undesired because the entry becomes permanently unusable). This may be difficult to do because it may be difficult to locate the capability and any copies of it that may have been made. A capability containing a unique code normally is revoked simply by unassigning the unique code, that is, by removing its translation from the hashing table. A unique code that has been unassigned is never reassigned. - Thus, a pointer-type identifier has the advantage of mapping efficiently to the address of the identified object but the disadvantage of making its containing capability difficult to revoke. unique code has the disadvantage of mapping less efficiently to the address of the identified object but the advantage of making its containing capability easy to revoke. - The method described here involves the use of a capability containing both a pointer-type identifier and a unique code. The pointer-type identifier is used in the normal way to determine the address of the identified object. The unique code is used as a validation number VN) to determine whether or not the capability is valid (has not been revoked). If the pointer-type identifier is a direct address, the VN is placed also in the identified object. If the pointer-type identifier is an indirect address, the VN is placed also in the entry in the intermediate mapping table. When the capability is used to access the object, the VN in the capability is compared to the VN in either the object or the intermediate table. The access is permitted only if the two compared VNs are equal; otherwise, an exception is recognized, and the access is prevented. The capability is revoked simply by changing (normally, incrementing by one) the VN in the object or the intermediate table. This revocation is done without accessing the capability. - The VNs used at different times with a particular pointer-type identifier need be unique only with respect to themselves, not with respect to the VNs used with other

pointer-type identifiers. This is advantageous because it reduces the number of bits needed to form a VN. - Different capabilities that identify the same object can be revoked selectively, as follows. If an attempt is made to use a capability whose VN is not equal to the VN in the object or the intermediate mapping table, and If that particular capability is not to have been revoked, the exception handling process can make the capability valid again by changing its VN so it is again equal to the VN in the object or the intermediate table. - An example of this method can be based upon the dual address space (DAS) facility of the IBM 3033 extension feature, as follows. - DAS provides a set of address spaces. Each address space is identified by an address-space number (ASN). DAS provides an ASN translation process that maps, by means of a two-level table lookup, an ASN to an ASN second table entry (ASTE). The ASTE contains a segment table designation (STD) that is used by the dynamic address translation (DAT) process to translate references to the address space corresponding to the ASTE. - In this example, a capability enables access to an address space. The capability contains the ASN of the address space. The ASN is considered here to be a pointer type identifier. The ASTE for the address space contains a VN. If the capability is valid, this same VN is also in the capability. - A capability is unforgeable in that it exists in storage that is protected by means of key-controlled protection from the process that uses the capability. - A process may use a new instruction to form, subject to appropriate authorization checks, a capability in the protected storage. This instruction has an ASN as an input operand. The instruction places the ASN in the capability, it finds the corresponding ASTE, and it copies the VN from the ASTE to the capability. The process may use new instructions to move the capability within the protected storage or to make copies of the capability within the protected storage. Thus, the control program, which subsequently may have reason to revoke the capability. - A process attempts to access an area in an address space by specifying the address of the area and by designating a capability that enables access to the address space. The manner in which the process designates the capability is not described here. As a result of this attempt, the machine obtains the ASN from the capability, uses this ASN to find the corresponding ASTE, and compares the VN in the capability to the VN in the ASTE. If the two VNs are equal, the machine completes the access.

L9: Entry 10 of 13 File: DWPI Sep 30, 1981

DERWENT-ACC-NO: 1981-K2824D
DERWENT-WEEK: 198140

TITLE: Data processing system having information protection - uses capability addressing which has access code resource type indicator in capability pointer

INVENTOR: HAMERHODGE, K J

PATENT-ASSIGNEE: PLESSEY CO LTD (PLES)

PRIORITY-DATA: 1977GB-0018616 (May 4, 1977)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
GB 1599421 A	September 30, 1981		020	

INT-CL (IPC): G06F 11/00

ABSTRACTED-PUB-NO: GB 1599421A
BASIC-ABSTRACT:

The data processing system has a central memory and at least one processor provided with a system capability table having one entry for each resource storing descriptor information. This facility indicates the location of that resource. Each processor is also provided with at least one reserved segment pointer table, which comprises a list of the capability pointers. The pointers have a resource type indicator, an access code and a pointer value.

The resource indicator defines an inform resource and an outform resource or a passive resource. When a capability pointer with a passive resource indicator is detected when a load capability register operation is in progress then the capability register to be loaded performs at an automatic trap operation, through a passive resource table. The descriptor for the latter is held in one of the hidden capability registers.

ABSTRACTED-PUB-NO: GB 1599421A
EQUIVALENT-ABSTRACTS:

DERWENT-CLASS: T01
EPI-CODES: T01-G01;

L4: Entry 6 of 15 File: USPT Nov 12, 2002

DOCUMENT-IDENTIFIER: US 6480845 B1
TITLE: Method and data processing system for emulating virtual memory working spaces

Detailed Description Text (5):
The word size in these systems is 36 bit. An archaic 6-bit code set is packed 6 alphanumeric characters per word. (ANSI 8-bit code has long been supported and is stored with a ninth padding bit at four characters to the word. A single 72 bit double length AQ register is separately accessible as the Arithmetic register and the Quotient register by various instructions. There are 8 index registers, augmented with 8 "address" registers employed with the string instructions. Finally, 8 "descriptor" registers hold capability tokens delimiting segments of virtual address space. The most commonly used of these 72 bit tokens provide both a base address (2**34-1 bytes) and a bound (up to 1 MB) plus flags for read, write, and other "permissions". Standard segments may begin on any byte and be as small as one byte long. The base can be in an of 512 "Working Spaces" that subdivide the 8 terabyte virtual address space. There are also five process definition descriptor registers including one that frames the current instruction segment. The description above is for illustrative purposes only, and is thus a simplification of the reality.

L4: Entry 9 of 15 File: USPT Dec 14, 1999

DOCUMENT-IDENTIFIER: US 6003123 A
TITLE: Memory system with global address translation

Detailed Description Text (3):
Guarded pointers are provided as a solution to the problem of providing efficient protection and sharing of data. Guarded pointers encode permission and segmentation information within tagged pointer objects to implement capability requirements of the type presented in Fabry, R., "Capability-based addressing," Communications of the ACM 17,7 (July 1974), 403-412. A guarded pointer may be stored in a general purpose register or in memory, eliminating the need for special storage. Because memory may be accessed directly using guarded pointers, higher performance may be achieved than with traditional implementations of capabilities, as table lookups to translate capabilities to virtual addresses are not required.

Detailed Description Text (6):
Memory systems that use guarded pointers provide a single virtual address space shared by all processes. Each guarded pointer identifies a segment of this address space that may be any power of two bytes in length, and must be aligned on its size. These restrictions allow six bits of segment length information and 54 bits of virtual address to completely specify a segment. The length field of a guarded pointer encodes the base-two logarithm of the segment length. This allows segments ranging in length from a single byte up to the entire 2.sup.54 byte address space in power of two increments. As shown in FIG. 1 the length field also divides the address into segment identifier and offset fields. A four-bit permission field completes the capability by identifying the set of operations permitted on the segment.

Detailed Description Text (60):
It is impossible in any capability-based system to directly revoke a single process' rights to access a segment without potentially affecting other processes. Since possession of a capability confers access rights, the only way to remove access rights from a single process is to remove all capabilities containing those access rights from the memory addressable by the process. This can be accomplished by sweeping the memory that the process can address, and overwriting the correct capabilities, so long as none of the memory containing those capabilities is shared. If the pointers that need to be overwritten are contained within a shared segment, all processes which rely on the pointer will lose access privileges. This is due to the lack of a protected table that stores permission information on a per-process basis.

Detailed Description Text (73):
A guarded pointer is an unforgeable handle to a segment of memory. Each pointer is comprised of segment permission, length, base, and offset fields. The advent of 64-bit machines allows this information to be encoded directly in a single word, without unduly limiting the memory address space. An additional tag bit is provided to prevent a user from illicitly creating a guarded pointer. Guarded pointers are an efficient implementation of capabilities without capability tables and mandatory indirection on memory access.

L29: Entry 18 of 37 File: USPT Nov 3, 1998

DOCUMENT-IDENTIFIER: US 5832299 A
TITLE: System for emulating input/output devices utilizing processor with virtual system mode by allowing mode interpreters to operate concurrently on different segment registers

Detailed Description Text (26):
Returning to FIG. 4, the segment descriptor register contains the segment descriptor 400 corresponding to the particular segment selector 302 loaded in a segment register (220-226). The segment descriptor 400 consists of a segment base address 402, a segment limit 404, and segment attribute and access right bits 406. The segment descriptor 400 contains two access bits, called DPL or descriptor privilege level bits, which define the least privileged level at which a task may access that descriptor, and an attribute bit termed the D-bit which indicates the default length for operands and offsets. If D=1 then 32-bit operands and 32-bit addressing modes are assumed. If D=0 then 16-bit operands and addressing modes are assumed. Regardless of the default precision of the operands or addresses, the processor is able to execute either 16-bit or 32-bit instructions by specifying an override prefix. The segment base address 402 defines the starting address of the segment. The segment limit 404 is combined with the granularity (G) bit to define a logical page size of 1 byte or 4 kbytes with an offset limit of 1 MB or 4 GB. The AVL, P, S, and A bits are used as in the 486 or Pentium and are described below.

L18: Entry 7 of 19 File: USPT Jan 2, 1996

DOCUMENT-IDENTIFIER: US 5481688 A
TITLE: Information processing system having an address translation table loaded with main/expanded memory presence bits

Detailed Description Text (7):
Turning to FIGS. 4A and 4B, each of the segment descriptor registers 206 stores segment descriptor defining segments which are logical data blocks. The segment descriptor comprises Base and Bound fields indicative of location and a size in a virtual space of the segment, respectively, and a W field indicative of a virtual space register number of a virtual space register (WSR) for storing a virtual space number (WSN). The segment descriptor further includes a Flag field indicative of an attribute of the segment. The attribute of the segment is shown in FIG. 4B.

L18: Entry 11 of 19 File: USPT Sep 5, 1989

DOCUMENT-IDENTIFIER: US 4864493 A
TITLE: Instruction address producing unit capable of accessing an instruction segment of an extended size

Brief Summary Text (9):
In Japanese Unexamined Patent Publication No. Syo 60-241,135, namely, 241,135/1985, an address producing method is described by H. Takane so as to define an increased address space, or an increased segment size, of each instruction segment by the use of a specific segment descriptor. Such a specific segment descriptor has a base field of 36 bits for the reference address of the specific segment descriptor, a bound field of 20 bits, and a mode indication bit representative of either a non-extension or normal mode or an extension mode. The specific segment descriptor is used as an extended segment descriptor when the extension mode is indicated by the mode indication bit. Otherwise, the specific segment descriptor is used as a normal segment descriptor.

Brief Summary Text (10):
In the extended mode, an additional bit set of 12 bits is combined with the bound field of the specific segment descriptor into an extended bound field of 32 bits. If each bit of the extended bound field is made to correspond to a single byte, the extended bound field of 2.sup.32 bits can define an extended segment of 2.sup.32 bytes.

Detailed Description Text (8):
The specified instruction descriptor has a base field 32 of 36 bits, a bound field 33 of 20 bits, and a control field of 8 bits. The control field is not shown in FIG. 2. The base field 32 specifies the instruction segment 16 in question by indicating a base or reference instruction address of the instruction segment 16. The bound field 33 defines a segment size of the instruction segment 16 and serves to determine whether or not the instruction segment is to be changed to another instruction segment. For this purpose, the bound address is always monitored in the known manner. Such a monitoring operation has no concern with the instant invention and therefore will not be described any longer

Detailed Description Text (11):
With this structure, the segment size of each instruction segment is restricted by the bound field 33 of the specified instruction descriptor and is therefore equal to or smaller than 2.sup.20 bytes.

Detailed Description Text (12):
As described in the preamble of the instant specification, consideration is directed to an extended instruction segment descriptor in the referenced Takane patent publication so as to extend a segment size of each instruction segment, such as 16. The extended instruction segment descriptor has a bound field of 20 bits and a base field of 32 bits like the instruction segment descriptor which is mentioned above and which may be called a normal instruction segment descriptor. The extended instruction segment descriptor is distinguished from the normal instruction segment descriptor in the processing unit 11 in a manner mentioned in the Takane patent publication and is kept in the instruction segment descriptor register 31. On reception of the extended instruction segment descriptor, the bound field of 20 bits is concatenated to an additional bit set of 12 bits and is processed as an extended bound field of 32 bits.

Detailed Description Text (19):
In FIG. 3, the primary segment descriptor which is read out of the segment descriptor segment 48 is indicated at 51 in FIG. 3. The primary segment descriptor 51 has a bound field of 20 bits, a base field of 36 bits following the bound field, and the control field of 8 bits depicted at a broken line and is kept in an instruction segment register 52. The instruction segment register 52 may be called a storing element and has a base area 53 for the base field of the primary segment descriptor and a bound area 54 for the bound field thereof.

Detailed Description Text (20):
Likewise, the additional segment descriptor read out of the segment descriptor segment 48 is depicted at 56 and has a base field of 36 bits, a bound field of 20 bits, and the control field. As to the additional segment descriptor 56, the base field of 36 bits may be considered with the bound and the control fields left out of consideration, as will become clear as the description proceeds.

Detailed Description Text (23):
More particularly, the bound field of the primary segment descriptor 51 is sent from the instruction segment register 52 to the bound extension circuit 61 which is controlled by the control field of the primary segment descriptor 51 in a manner similar to that described in the referenced Takane patent publication. In the extension mode, the bound field of 20 bits and an additional bit set of 12 bits are concatenated together into an extended bound field of 32 bits in the bound extension circuit 61 as mentioned in the referenced publication. Consequently, the instruction segment 59 can have the segment size equal in number to 2.sup.32 bytes at maximum.

Detailed Description Text (32):
The primary segment descriptor 51 is stored in the instruction segment register 52 in the manner described in FIG. 3. The bound field of 36 bits is

Detailed Description Text (33):
restricted to 32 bits by removing four lower bits from the bound field of the primary segment descriptor 51. The first retained instruction descriptor of 32 bits is delivered to the adder circuit 58.

Detailed Description Text (34):
As shown in FIG. 5, the bound field of the additional segment descriptor 56 is sent to the instruction counter 57 of 32 bits. In this event, thirty-two upper bits of the bound field of the additional segment descriptor 56 are kept in the instruction counter 57 to be sent to the adder circuit 58 as the second retained instruction descriptor of 32 bits. The result of addition is sent to the memory unit 10 as the instruction address.

CLAIMS:

2. An instruction address producing unit as claimed in claim 1, said instruction address being for indicating an instruction which is located in said instruction descriptor segment having a base address and a predetermined size, each of said first and said second instruction segment descriptors having a base field for specifying said base address while said first instruction segment descriptor has a bound field for specifying said predetermined size, wherein said retaining means comprises:

storing means coupled to said memory unit for storing said first instruction segment descriptor; and

instruction counter means for loading the base field of said second instruction segment descriptor;

said instruction address deriving means being coupled to said storing means and said instruction counter means and comprising:

adding means coupled to said storing means and said instruction counter means for adding the base field of said first instruction segment descriptor to the base field of said second instruction segment descriptor to produce said instruction address.

WEST

Generate Collection Print

L25: Entry 1 of 14 File: USPT Nov 12, 2002

DOCUMENT-IDENTIFIER: US 6480845 B1
TITLE: Method and data processing system for emulating virtual memory working spaces

Detailed Description Text (5):
The word size in these systems is 36 bit. An archaic 6-bit code set is packed 6 alphanumeric characters per word. (ANSI 8-bit code has long been supported and is stored with a ninth padding bit at four characters to the word. A single 72 bit double length AQ register is separately accessible as the Arithmetic register and the Quotient register by various instructions. There are 8 index registers, augmented with 8 "address" registers employed with the string instructions. Finally, 8 "descriptor" registers hold capability tokens delimiting segments of virtual address space. The most commonly used of these 72 bit tokens provide both a base address (2**34-1 bytes) and a bound (up to 1 MB) plus flags for read, write, and other "permissions". Standard segments may begin on any byte and be as small as one byte long. The base can be in an of 512 "Working Spaces" that subdivide the 8 terabyte virtual address space. There are also five process definition descriptor registers including one that frames the current instruction segment. The description above is for illustrative purposes only, and is thus a simplification of the reality.